Paralelización de algoritmos de Sort-Merge sobre distintas arquitecturas. Evaluación de performance.

Laura De Giusti[1], Diego Tarrio[1], Ing. A.De Giusti[2], Lic. Marcelo Naiouf [3]

Laboratorio de Investigación y Desarrollo en Informática[4]
Departamento de Informática - Facultad de Ciencias Exactas
Universidad Nacional de La Plata

Resumen

Se presenta el desarrollo y análisis de performance de la paralelización de un algoritmo de ordenación de archivos basado en la técnica de Sort-Merge sobre distintas arquitecturas.

Se estudian tres modelos de arquitectura: monoprocesador, red heterogénea con soporte PVM e hipercubo de transputers , trabajando en C y C paralelo.

El algoritmo planteado consiste en una paralelización funcional y de datos del método de ordenación Merge-Sort, definiendo un conjunto de procesos encargados de la ordenación y mezcla de bloques de archivos, coordinados por un proceso maestro.

La metodología del trabajo consistió en las siguientes etapas:

1. La implementación del algoritmo sobre una arquitectura de red con procesadores heterogéneos.
2. Migración del algoritmo a una arquitectura paralela (hipercubo de transputers).
3. Obtención de tiempos de ejecución, comunicación y sincronismo para su posterior comparación.

Por último, se analizan resultados y conclusiones obtenidas considerando factores de tiempo, performance, speed up y escalabilidad al aplicar el algoritmo en cada arquitectura.

Palabras clave: Procesamiento distribuido y paralelo. Performance.

[1] Alumnos avanzados de la Licenciatura en Informática. Dpto de Informática, Fac. de Ciencias Exactas, UNLP. Becarios Alumnos L.I.D.I.
E-mail: {ldgiusti, dtarrio}@info.unlp.edu.ar

[2] Inv. Principal CONICET. Profesor Tit. Ded. Excl., Dpto. de Informática, Facultad de Cs. Exactas, UNLP.
E-mail: degiusti@info.unlp.edu.ar

[3] Profesor Adjunto Ded. Excl., Dpto. de Informática, Facultad de Cs. Exactas, UNLP. E-mail: mnaiouf@info.unlp.edu.ar

[4] Calle 50 y 115 Primer Piso, (1900) La Plata, Argentina, Teléfono 54-21-227707
WEB: lidi.info.unlp.edu.ar

Introducción

Históricamente, la única forma de tratar algunos problemas de procesamiento ha sido por medio de cómputo paralelo [Heer91] [Tine98]. Existen muchas aplicaciones en las que se necesita procesar grandes cantidades de datos en muy poco tiempo. Por ejemplo, los satélites meteorológicos recogen datos de la tierra a razón de 10^{10} bits por segundo y para poder utilizar esta información se debe procesarla al menos a una velocidad de 10^{13} operaciones por segundo; las aplicaciones médicas, y particularmente las relacionadas con cirugía necesitan una velocidad de procesamiento mínima de 10^{15} operaciones por segundo; los controles de vuelo y los controles de armas requieren capacidad de procesamiento en tiempo real del orden de 10^{18} operaciones por segundo.

A pesar de los avances en términos de velocidad de procesamiento que se han obtenido en los procesadores, es evidente que hay un límite físico dado por la velocidad de la luz: aunque dentro de un componente electrónico se puedan realizar operaciones muy rápidamente, si se necesita comunicar con otro componente, el tiempo de comunicación de las señales está limitado y se pierde la ganancia de velocidad.

Por lo tanto, la única forma de tratar algunos problemas es la utilización de procesamiento paralelo [Leig92] [Mors94] [Laws92] [Coff92]. Si varias operaciones pueden ser ejecutadas simultáneamente, el tiempo total de procesamiento se verá reducido, aún cuando cada una de las operaciones no se lleve a cabo más rápidamente. Tradicionalmente, el término *supercomputadora* estuvo relacionado de una manera u otra con *procesamiento paralelo*. Al mismo tiempo la especificación de algoritmos ejecutables sobre una arquitectura paralela, ha estado asociada por la *programación concurrente*. [Hoar85] [Andr91] [Burn93].

Se han propuesto múltiples formas de organizar las arquitecturas de procesamiento paralelo, e incluso se considera un problema definir qué significa de manera precisa una "arquitectura paralela" [Hwan93] [Code92]. El problema de definir qué es, y luego dar una taxonomía de las arquitecturas de procesamiento paralelo, se encuentra en la gran cantidad de características que se tienen en una computadora paralela y que no todas son de fácil descripción, comparación y clasificación.

La taxonomía clásica de las arquitecturas de procesamiento establecida inicialmente en [Fly72] se centra en la forma en que se ejecutan las instrucciones sobre los datos:

- SISD: **S**ingle **I**nstruction . **S**ingle **D**ata .
- MISD: **M**ultiple **I**nstruction . **S**ingle **D**ata .
- SIMD: **S**ingle **I**nstruction . **M**ultiple **D**ata.
- MIMD: **M**ultiple **I**nstruction . **M**ultiple **D**ata.

Para nuestro interés, nos concentraremos en las arquitecturas genéricamente conocidas como MIMD, tanto de granularidad gruesa (por ejemplo, las redes de máquinas heterogéneas) como de granularidad fina (los hipercubos de pequeños procesadores homogéneos) [Niel90] [Stee96] [DayK94] [Kung90] [Kung91].

La Figura 1 muestra esquemáticamente una arquitectura MIMD genérica, con memoria distribuida. En línea punteada se marca la posibilidad de agregar una memoria compartida.

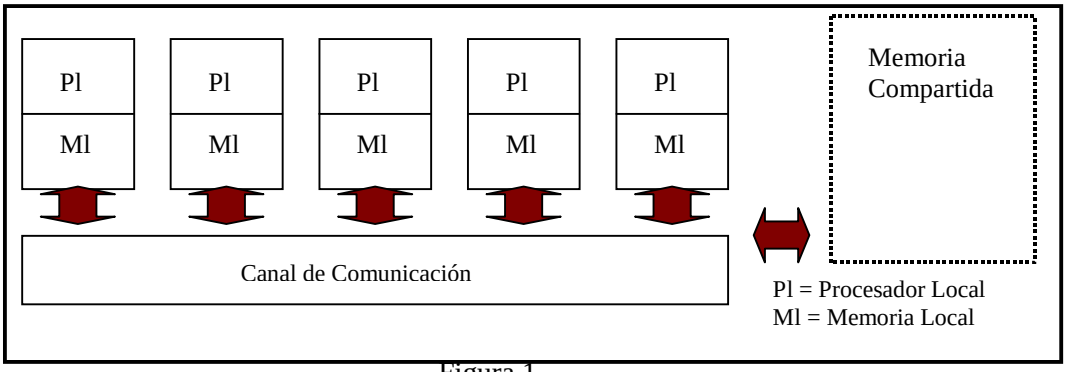


Figura 1

El procesador (P) equivale a la unidad de control (CU) más la unidad de procesamiento (PU). Los streams de datos y de instrucciones (DS y DI) de cada procesador provienen de su propia memoria. A través de los links de comunicaciones más la red de interconexión se pueden enviar mensajes a los demás procesadores. Se podría definir como múltiples SISD interconectadas.

Nótese que en el caso de trabajar con una arquitectura de grano grueso el subsistema de comunicaciones puede ser por ejemplo un bus compartido (local y/o remoto) y la capacidad de cada procesador local equivaler a una Workstation con su propio sistema operativo; en cambio, si se trabaja con un hipercubo de transputers por ejemplo, la capacidad local será mucho menor pero el subsistema de comunicaciones resolverá simultáneamente, por hardware y a muy alta velocidad la vinculación entre los procesos residentes en diferentes procesadores. Estas diferencias hacen particularmente interesante investigar la implantación de algoritmos paralelos clásicos [Colb96] [WuIC91] [Niga95] [Mill98] sobre arquitecturas MIMD de diferente granularidad, buscando evaluar parámetros de performance del cómputo paralelo [Gupt93] [Gust92] [SunX90] [Buba97] [SunX95].

En este trabajo, desarrollado básicamente por alumnos que se inician en la investigación en el LIDI, se analiza el tratamiento paralelo de datos distribuidos sobre N procesadores, ordenándolos por el método de Sort-Merge buscando obtener parámetros de performance, así como experiencia en la migración de algoritmos secuenciales a los dos modelos extremos de arquitectura MIMD que se plantean. Para ello se ha utilizado una red convencional con soporte PVM y un hipercubo de 32 transputers disponible en el Laboratorio.

Modelos de Arquitectura a estudiar

1-Red heterogénea con soporte Pvm:PVM (Parallel Virtual Machine) es un paquete de software con las siguientes características [Dong94]:

- Provee protocolos para la comunicación entre una colección de computadoras heterogéneas conectadas a través de una red, haciendo que esta se vea como una única máquina virtual paralela.
- Permite a los usuarios resolver problemas computacionalmente complejos, con un costo adicional mínimo combinando la potencia de cálculo y memoria de varios computadores.

- Utiliza el modelo de pasaje de mensajes asincrónico permitiendo a los programadores explotar el procesamiento distribuido a través de una amplia variedad de tipos de computadores. El direccionamiento de los mensajes y la conversión de datos se realiza en forma transparente al programador produciendo un software altamente portable (figura 2).
- Puede planificar la distribución de los procesos a través de los distintos procesadores balanceando la carga de los mismos.
- El modelo de computación con PVM se basa en la noción que una aplicación consiste en varias tareas. La división de tareas se basa en un paralelismo funcional o en un paralelismo de datos.

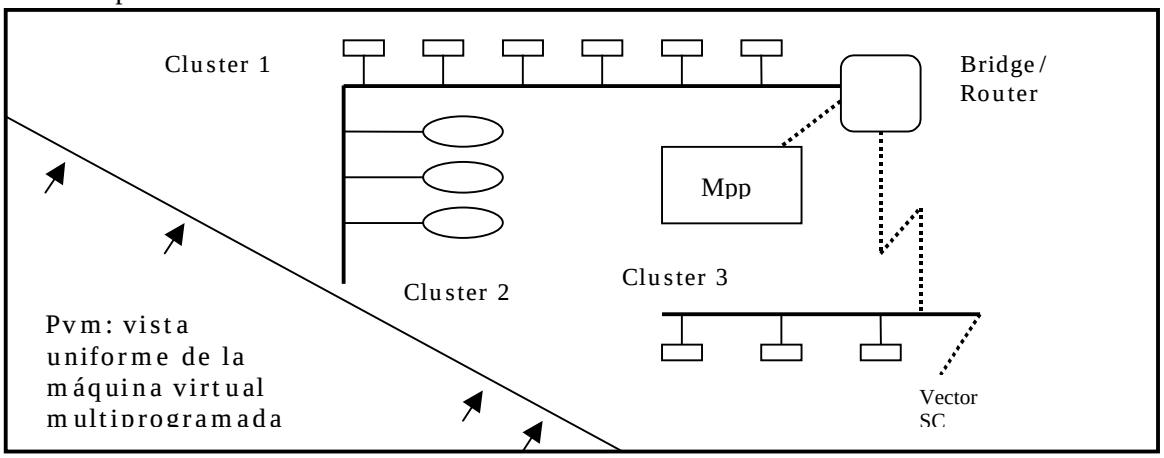


Figura 2

2-Hipercubo de Transputers:

Transputer: Se trata de un procesador tipo RISC (Repertorio de Instrucciones Reducido) constituido por los siguientes bloques: CPU, links, eventos, memoria, interface de memoria externa, reloj y otros servicios (Figura 3), pudiendo adicionar una unidad de punto flotante (FPU) o una unidad de procesamiento de disco [CSA90][Logi94].

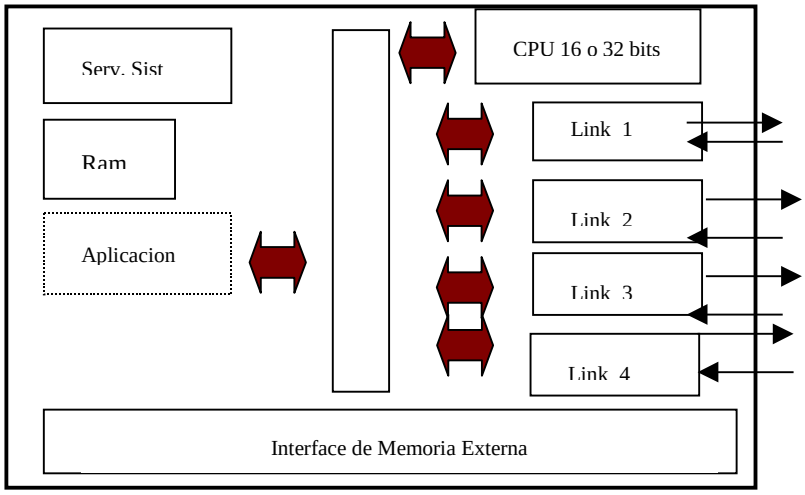


Figura 3

Links: Un transputer posee 4 links permitiendo conectarse con un máximo de 4 transputers a través de los mismos. Cada link posee dos controladores de tipo DMA (Acceso Directo a Memoria), uno para entrada y otro para salida. Cada uno de ellos puede leer o escribir en

memoria externa o interna sin la intervención de la CPU, permitiendo comunicaciones en paralelo . Cada link provee dos canales, uno en cada dirección, de esta manera un cable transporta datos por un canal y control (ACKs) por el otro, permitiendo la comunicación bidireccional entre 2 transputers (Figura 4).

En ciertas aplicaciones es posible la necesidad de comunicación entre procesos alocados en transputers no adyacentes, una solución a esto consiste en la utilización de canales virtuales definidos por software.

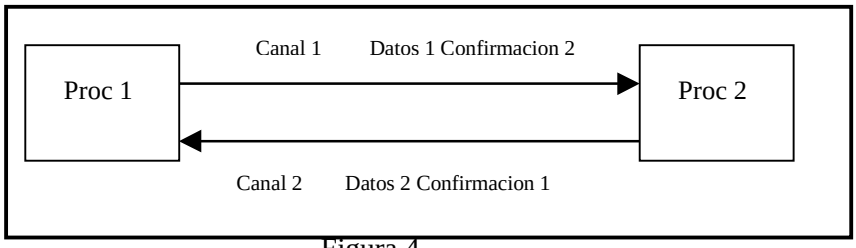


Figura 4

Eventos: Las interrupciones son comunicaciones simples sin datos, solo proveen sincronismo. Un pedido de interrupción (EventReq) se asiente (EventAck) si el proceso residente en el transputer esta listo para proceder, también se puede producir una espera de interrupción (EventWaiting).

Interface de memoria externa: Los transputers poseen una interface de memoria externa multiplexada con soporte externo de DMA.

Hipercubo (definición): Un d-cubo está compuesto por 2^d procesadores dispersos en **d** dimensiones, con **2** procesadores por dimensión y donde cada procesador está conectado a otros d procesadores (figura 5):

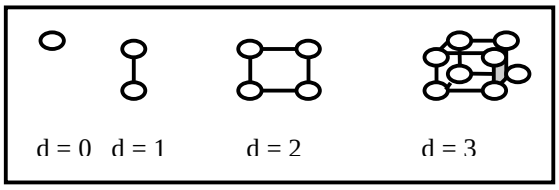


Figura 5

Cuando un d-cubo tiene más de tres dimensiones se lo denomina **hipercubo.** La dimensión de un hipercubo está definida de la siguiente manera **d** = log p (donde p es la cantidad de procesadores). El **grado** de un hipercubo es igual a la dimensión.

En un hipercubo los procesadores son numerados del 0 al $(2^d - 1)$.

Las características generales de los hipercubos son :

- Puede ser **definido recursivamente**, esto es, un hipercubo de dimensión cero conecta un solo procesador, y un hipercubo de dimensión uno conecta dos hipercubos de dimensión cero.
- Es una de las topologías más versátiles y eficientes para el procesamiento paralelo.
- Puede ser adaptada para tareas de propósito general, así como también para la resolución de problemas específicos (Ejemplo: Transformada de Fourier).
- Puede simular de manera eficiente cualquier otra topología del mismo tamaño. Un hipercubo de p procesadores puede simular por ejemplo :
    1. Arreglo de p procesadores.
    2. Arbol binario de p procesadores
    3. Malla, Tours, etc.

- Una de las desventajas principales consiste en que el número de conexiones de cada procesador crece de manera logarítmica con el tamaño de la red. Esto quiere decir que al aumentar la cantidad de procesadores se incrementan las conexiones y el overhead de comunicaciones asociado .
- El número de procesadores debe ser una potencia de 2; esto constituye una limitación debido al elevado costo de crecimiento de la dimensión del hipercubo.

Cada uno de los procesadores que conforman un hipercubo puede trabajar de dos maneras:

- Cada procesador trabaja en forma separada, esto se debe a que contiene su propio código de instrucciones con sus propios datos.
- Los procesadores cooperan entre sí, comunicándose por medio mensajes para compartir sus datos.

Para este trabajo se utilizó un hipercubo de dimensión 5 (constituido por 32 transputers de la serie T800 de 32 bits), cuya estructura se basa en una PC Pentium con una placa Ultra/XL^{tm} que conecta a un grupo de 8 placas VME-XP^{TM} con 4 transputers cada una. La placa Ultra/XL^{tm} dispone de un único transputer **host** sin capacidad de procesamiento, es decir, solo se encarga de las operaciones de entrada/salida del nodo 0.

Algoritmo de Sort-Merge. Paralelización.

1-Algoritmo Sort-Merge: El algoritmo que se plantea en esta experiencia es el siguiente: Se tiene un conjunto de datos almacenados en un archivo desordenado de tamaño considerable (10 Mbytes a 1Gbyte en las experiencias realizadas). Un paso inicial es hacer M divisiones del archivo original. Luego con cada una de esas M divisiones se crean M subarchivos desordenados. Un tercer paso consiste en ordenar cada uno de estos M subarchivos en forma separada por medio de cualquier método de ordenación (se aplicó el método Quick-Sort). Como último paso se toman los M subarchivos ordenados y se realiza un merge obteniendo el archivo final ordenado (Figura 6).

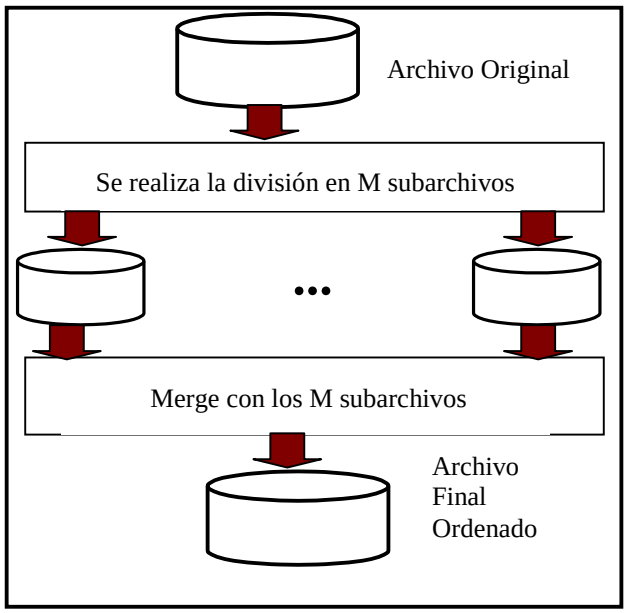


Figura 6

Nótese que un algoritmo puede ser paralelizado teniendo en cuenta dos aspectos, un primer aspecto es la paralelización de datos (en donde se generan varias instancias de una misma tarea y cada una de ellas procesa los datos en forma separada pero en forma paralela); el otro es la paralelización funcional (en donde cada tarea ejecuta una función diferente).

El algoritmo fue paralelizado teniendo en cuenta los dos aspectos descriptos anteriormente. El trabajo realizado sobre cada subarchivo es independiente entre sí, obteniendo así un paralelismo de datos a través de un modelo "master-slave", donde cada instancia de una misma tarea procesa una porción del archivo.

En cuanto a la paralelización funcional se definen distintas tareas donde cada una realiza una función distinta (División,Ordenación,Mezcla).

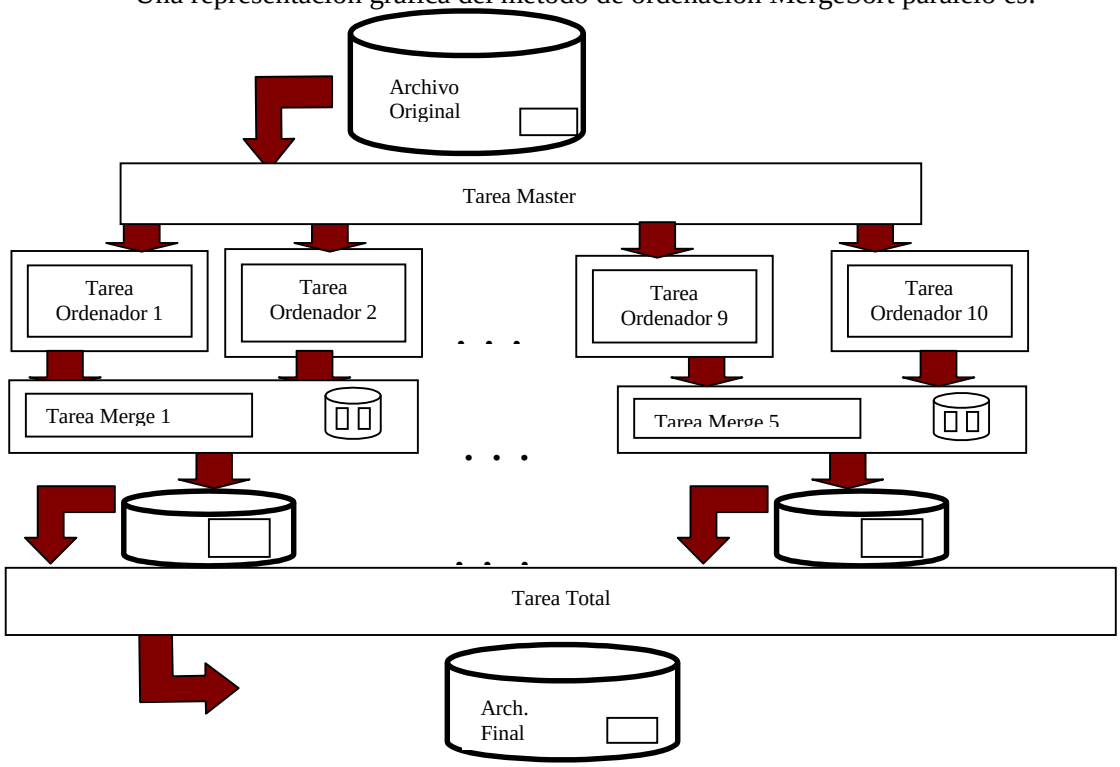
Se paralelizó el algoritmo de la siguiente manera : Primero el archivo original se divide en M porciones de N MBytes cada una. Para realizar esta función se crea un tarea "Master" que será la encargada de leer los M bloques . Cada uno de estos bloques se asigna a una tarea que realiza la ordenación del mismo utilizando el algoritmo QuickSort, por lo tanto se crean M instancias de la tarea "Ordenador". Cada K tareas Ordenador se crea una instancia de la tarea "Merge" que se encarga de recibir K porciones ordenadas del archivo, intercalándolas generando una porción de K*N MBytes ordenada la cual es enviada a la tarea "Total". Existirá una única tarea "Total" la cual es la encargada de intercalar las porciones recibidas de las M/K tareas "Merge", generando el archivo final ordenado.

1a- Paralelización sobre la arquitectura MIMD con PVM: Teniendo en cuenta el tipo de arquitectura (donde cada procesador dispone de su propia memoria y disco), se hizo posible paralelizar los accesos a disco (en especial los desplazamientos, factor crítico en el tiempo de ejecución del algoritmo).

Una prueba del algoritmo se efectuó sobre un archivo de 100.000 registros de 100 bytes cada uno y una red heterogénea linux (donde cada procesador dispone de 1 Mb de memoria RAM para almacenamiento de datos). Basándonos en las tareas explicadas anteriormente y sus funciones se implementó el algoritmo según los siguientes parámetros:

M= 10 (Nro. de porciones y de tareas "ordenador").
N= 1Mb (Tamaño de cada porción del archivo).
K= 2 (Nro de porciones ordenadas asignadas a cada tarea "merge").
M/K = 5 (Cantidad de tareas "merge").

Una representación gráfica del método de ordenación MergeSort paralelo es:



Para analizar el orden de ejecución de este algoritmo se considera el “Pipe Line” constituido por una única rama del gráfico anterior (Figura 7).

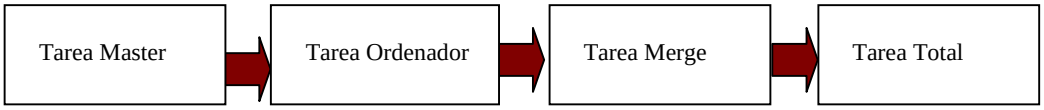


Figura 7

Por lo tanto, el orden de ejecución del algoritmo paralelo es O(N (log N)), Debido al elevado número de accesos a disco realizados por el algoritmo, centralizaremos el análisis principalmente en el número de desplazamientos efectuados en disco:

a- Cada tarea “Merge” intercala 2 archivos de 10.000 registros cada uno dividiendo su área de datos en RAM en 2 porciones de 5.000 registros =>
Nro. Desplazamientos de "$Merge_i$" = 2 despl. * 2 archivos = 4.

b- La tarea "Total" recibe 5 archivos de 20.000 registros =>
divide su área de datos en RAM en 5 porciones de 2.000 registros c/u.
Nro. Desplaz. "Total"= 10 desplaz. * 5 archivos = 50.

∴ Nro. Total de desplazamientos = (4 + 50 = 54).

1b- Implementación sobre el hipercubo de transputers: Teniendo en cuenta que uno de los objetivos principales de este estudio era la evaluación de performance de las distintas arquitecturas se trabajó sobre el mismo algoritmo.

Desde un principio se observó la necesidad de redefinir la topología del hipercubo para adecuarla al algoritmo Merge-Sort. Esta necesidad surge cuando la tarea Master se comunica con M tareas Ordenador, haciendo necesaria la utilización de canales virtuales ante la limitación de los cuatro canales físicos de los transputers, esta decisión incrementa el overhead de comunicaciones y de memoria, cuyo análisis se detalla posteriormente al explicar los resultados obtenidos.

La migración del algoritmo desarrollado en Pvm al hipercubo consistió en la implementación de las mismas tareas asignando cada una de ellas a un transputer distinto.

Una de las **desventajas** observadas en esta arquitectura es la existencia de un único disco con el consecuente acceso secuencial al mismo impidiendo la paralelización de desplazamientos antes mencionada.

Para posibilitar el acceso concurrente a disco fue necesario utilizar un servidor residente (CIO server) provisto por el paquete de software del hipercubo de transputers.

Una dificultad observada al analizar el algoritmo en esta arquitectura se produce al intentar evaluar el overhead de comunicaciones y memoria debido a la utilización de canales virtuales asociados a una no vinculación física directa. El uso de estos canales hace necesario mantener información adicional de ruteo y control (que aunque se maneje en forma transparente al usuario incrementa el overhead en comunicación y memoria).

Una alternativa a la migración del problema de Ordenación de Archivos que se adapte en forma natural a la topología del hipercubo, consistió en implementar un algoritmo del tipo Odd-Even (el cual intercambia datos entre procesadores adyacentes utilizando solo los canales físicos provistos por los transputers). Esto puede representar una ventaja con respecto al algoritmo implementado al eliminar el overhead impuesto por la utilización de canales virtuales, pero esto

desviaría del objetivo inicial que era evaluar el mismo algoritmo en las distintas arquitecturas junto con el costo de migración asociado.

Métricas de paralelismo

Los factores de métricas a analizar son los siguientes:

1- Factores a tener en cuenta al analizar el desempeño del algoritmo paralelo: Muchas de las arquitecturas paralelas no alcanzan su capacidad teórica para diferentes aplicaciones, las causas de degradación de una máquina paralela se deben en gran medida a una falta de correspondencia entre aplicaciones, software y hardware (Figura 9).

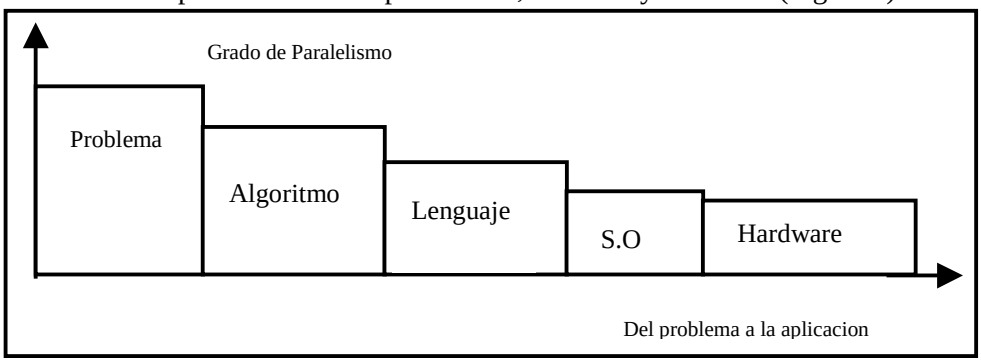


Figura 9

2- Factor de Aceleración (Speed up): Sea p el número de procesadores, Ts(N) el tiempo requerido para resolver el problema de tamaño N por el algoritmo secuencial y $T_p(N)$ el tiempo necesario para que el algoritmo paralelo resuelva el mismo problema usando p procesadores.

Entonces:

$$Sp <= \frac{Ts(N)}{T_p(N)}$$

$$Sp = \frac{\text{Tiempo ejecución secuencial}}{\text{Tiempo ejecución paralelo}}$$

$$1 <= Sp <= p$$

3- Análisis de pérdida de tiempo en operaciones de:

- Sincronismo.
- Overhead.
- Comunicación

4- Eficiencia: Se define eficiencia como el cociente dado por el factor de aceleración y el número de procesadores.

$$Ep = \frac{Sp}{p} \qquad 0 < Ep \leq 1$$

5- Redundancia: Se define redundancia como el cociente dado por el número total de operaciones realizadas usando p procesadores (Op) y el número de operaciones realizadas usando un procesador (O_1).

$$Rp = \frac{Op}{O1} \qquad Rp => 1$$

6- Ley de Amdahl: Permite obtener la aceleración máxima alcanzada usando p procesadores [Amda67]. Sea f la fracción de operaciones de un algoritmo que deben ser ejecutadas secuencialmente, donde $0 \leq f \leq 1$.

$$Sp <= \frac{1}{f+\frac{(1-f)}{p}}$$

Sobre esta métrica cabe citar a Gustafsson que reevalúa la ley de Amdahl observando que no considera la escalabilidad en el cálculo del speedup [Gust88]. A los fines de este trabajo valen las consideraciones básicas de Amdahl respecto de una asíntota máxima para el crecimiento del speedup.

Este trabajo hace hincapié en el análisis de speed-up, eficiencia, tiempos requeridos para la comunicación y sincronismo, y el overhead asociado a la comunicación entre procesos.

Aspectos de Implementación

A continuación se citan las herramientas (lenguajes, cantidad de procesadores, tamaño del archivo, etc) utilizadas para el análisis de resultados :

Tamaño de archivos: en una de las pruebas se utilizó un archivo de 10Mbytes compuesto por registros de 100 bytes cada uno.

Cantidad de procesadores: se ejecutó el algoritmo sobre un número variable de procesadores en ambas arquitecturas para poder analizar el factor de speedup según la cantidad de procesadores.[1 a 32].

Lenguaje: Se implementó el algoritmo en el lenguaje C, utilizando librerías específicas para el soporte de comunicaciones en la red heterogénea (pvm.h) y para el tratamiento de los canales en el hipercubo de transputers (conc.h). Si bien el algoritmo sobre el hipercubo de transputers se pudo programar en OCCAM produciendo un código más compacto y legible, respetamos el objetivo principal de comparar ambas arquitecturas en condiciones similares.

Un aspecto importante en la implementación fue la posibilidad de poder separar el costo de comunicación entre los distintos procesos en la red heterogénea y en el hipercubo de transputers, del costo de ejecución del algoritmo.

Resultados obtenidos

1-Comunicaciones: Se tuvieron en cuenta las siguientes características al analizar los costos y overhead de comunicación en las distintas arquitecturas:

En la red heterogénea la comunicación proporcionada por PVM se realiza a través del Sistema Operativo, utilizando las primitivas pvm_send y pvm_recv, lo que implica un costo adicional de tiempo al depender de la planificación utilizada por el sistema subyacente. Recordando que desde el punto de vista del programador la red heterogénea se ve como una única máquina virtual paralela dos procesos que desean comunicarse, pueden necesitar conocer sus respectivas

identidades, utilizando las primitivas pvm_parent y pvm_mytid, esta necesidad se observa cuando cada tarea "merge" necesita conocer la identidad de las dos tareas "ordenador" correspondientes, las cuales le envían sus respectivas porciones a intercalar. Cada tarea obtiene su identificación de proceso cuando la tarea "master" las crea dinámicamente mediante la primitiva pvm_spawn.

En el caso del hipercubo de transputers la comunicación está resuelta en la arquitectura del procesador mismo, lo que permite una comunicación más rápida y sin interrupciones a la CPU, utilizando las primitivas vchan_in y vchan_out. Cada canal tiene asociado un proceso emisor y uno receptor definido estáticamente en el archivo de configuración, esto elimina la necesidad de indicar en tiempo de ejecución la identificación de los nodos conectados. La desventaja de esta configuración estática está asociada a la dificultad de debugging y a ser propensa a errores.

2-Alocación de tareas:

En la red heterogénea el mecanismo de alocación de tareas proporcionado por PVM es dinámico, supervisado por un proceso residente en cada host de la red (PVM-daemon),

En el caso del hipercubo de transputers la alocación de tareas se define al configurar la red, por medio de un archivo de configuración; acercando al programador a la topología del hipercubo.

3-Mediciones:

Tiempos de comunicación y Overhead: Fue necesario un incremento en las comunicaciones sobre la red heterogénea para poder determinar a qué proceso se envía y/o recibe datos. El overhead de comunicaciones de la prueba efectuada se resume en la siguiente tabla:

	Master	Ordenadores * 10	Merge * 5	Total	Σ comunicaciones
Send Datos	10	1	2	---	30
Send Info	6	---	6	---	36
Receive Datos	---	1	2	10	30
Receive Info	---	1	1	21	36

Tabla 1.

$\therefore$ total de comunicaciones efectuadas entre procesos = 66.

Se observa que aproximadamente un 55 % de las comunicaciones entre los procesos a través de la red heterogénea con soporte PVM sólo transmite información correspondiente a identidad de procesos y/o tamaños de buffers a enviar. Teniendo en cuenta que el tiempo de comunicación promedio obtenido en nuestras experiencias es de 2.5 milisegundos y observando el pipeline del gráfico 7 obtenemos una cota inferior proporcional al número de comunicaciones secuenciales (29 siguiendo el pipeline) del tiempo de transmisión de información adicional a los datos cercano a los 29*2.5= 72.5 milisegundos .

En el hipercubo de transputers las comunicaciones necesarias sólo transmiten las porciones de archivos entre procesos, esto representa una ventaja con respecto al overhead de comunicaciones en la red heterogénea PVM debido a la eliminación de los send/receive de información utilizados en la red PVM. Esta ventaja se acentúa al considerar que la velocidad de transmisión a través de

canales físicos en los transputers se realiza a una velocidad del orden de μs. La desventaja principal en esta arquitectura se debe a la utilización de canales virtuales, que asignan a cada transputer del hipercubo tablas y buffers de ruteo produciendo el siguiente overhead de memoria por cada transputer:

Overhead de memoria x transputer = TTR + (TBR * # BR)

Siendo TTR= tamaño de la tabla de ruteo (fija para todos los nodos) ≈ N° nodos/4;
TBR = tamaño del buffer de ruteo ≈ 1024 bytes por default.
# BR = Número de buffers de ruteo por nodo (está acotado por el número de canales virtuales asignados al nodo).

El nodo que tiene asignada la tarea Master (Nodo_Master) es el que posee mayor cantidad de canales virtuales, por consiguiente se obtiene la siguiente cota superior de overhead de memoria para los nodos del hipercubo:

Overhead de memoria ≤ (32 / 4) + (1024*10)= 10248 bytes . Considerando que cada transputer dispone de 1Mb de memoria local, 10 Kb se pueden desperdiciar en el uso de canales virtuales.

Speed-up:

La solución monotarea, monoprocesador completó la ordenación en 24 segundos.

Las pruebas realizadas variando el número de procesadores sobre la red heterogénea con soporte PVM arrojó los siguientes resultados en tiempos de ejecución variando el número de procesadores (Nota: los tiempos están expresados en segundos):

Nº Hosts	1	2	3	4	5
Tiempo	42,5	22,7	18,3	16,5	12,7

Se observa que el tiempo adicional de la solución multitarea, monoprocesador incrementa el tiempo de ejecución del algoritmo secuencial en un 43,5 % debido a los tiempos de comunicación (30 envíos de bloques de 1Mb entre las tareas) y desplazamientos en disco adicionales debidos a la planificación de procesos del sistema operativo utilizado (Linux en nuestras experiencias).

Al incrementar el número de procesadores a 5 se alcanza un speedup cercano a 2 este resultado no se aproxima al speedup máximo de 5 debido principalmente al costo asociado a las comunicaciones en PVM (recordando la necesidad de empaquetar/desempaquetar los datos al enviar/recibir entre tareas).

- El archivo ordenado se obtiene sobre el hipercubo de transputers en un tiempo cercano a 15,5 segundos. Este resultado muestra que, si bien el costo de comunicación entre transputers es mucho menor comparado a la red heterogénea PVM permitiendo una capacidad de procesamiento superior en tiempo, el procesamiento de grandes volúmenes de datos sobre esta arquitectura tiene la desventaja del acceso secuencial a un único disco compartido, esto impide incrementar el speedup independientemente del número de procesadores utilizados, ante la imposibilidad de paralelizar los accesos a disco (figura 10).

Finalmente, se muestran algunos resultados del speed-up obtenidos en ambas arquitecturas variando el número de procesadores en la siguiente tabla y en la figura 10:

Nº procesadores	3	5	8	17
Speed-Up Pvm	1,3	1,9	2,2	2,4
Speed-Up Hipercubo	1,2	1,3	1,4	1,6

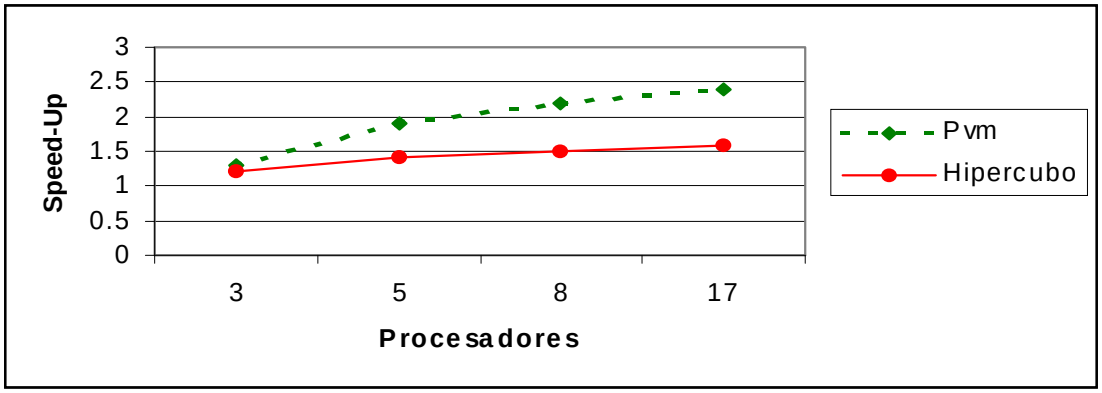


figura 10.

Líneas de trabajo:

Si bien los resultados obtenidos satisfacen los objetivos planteados, se comenzó a encarar líneas de trabajo alternativas con la intención de obtener mejoras en los siguientes aspectos:

- Investigación en paralelización de algoritmos secuenciales.
- Investigación en la separación del costo de comunicaciones respecto del costo propio de los algoritmos.
- Arquitecturas adaptivas.
- Comparación con arquitecturas adaptadas a clases de problemas (multi.DSP para tratamiento de imágenes).
- Bases de datos distribuidas y su administración en una arquitectura MIMD.

Bibliografía

[Amda67] Amdahl G., “On the Validty of the Single-Processor Approach to Achieving Large-SCALE Computing Capabilities”, Proc. Of the AFIPS Conference, 1967, pp. 483-485

[Andr91] Andrews G., “Concurrent Programming”, The Benjaming/Cummings Publishing Company, 1991.

[Buba97] Bubak M., Funika W., Moscinski J., “Performance Analysis of Parallel Applications under Message Passing Environments”, www.icsr.agh.edu.pl/publications/html/perf_full/perf_full.html, 1997

[Burn93] Burns A., Davies G., “Concurrent Programming”, Addison Wesley, 1993.

[Code92] Codenotti B., Leoncini M., “Introduction to Parallel Processing”, Addison Wesley, 1992.

[Coff92] M. Coffin, "Parallel programming- A new approach", Prentice Hall, Englewood Cliffs, 1992.

[Colb96] Colbrook A., Brewer E., Dellarocas C., Weihl W., “Algorithms for Search Trees on Message-Passing Architectures”, IEEE Transactions on Parallel and Distributed Systems, Vol. 7, No. 2: February 1996, pp. 97-108

[CSA90] “Transputer Architecture”, Computer System Architects, 1990.

[DayK94] Day K., Tripathi A., “A Comparative Study of Topological Properties of Hypercubes and Star Graphs” IEEE Trans. Parallel and Distributed Systems, vol. 5, no. 1, pp. 31-38, Jan. 1994

[Dong94] Dongarra J., et al, “PMV: Parallel Virtual Machine, A User´s Guide and Tutorial for Neworked Parallel Computing”, MIT Press, 1994.

[Flyn72] Flynn M. J., “Some Computer Organizations and Their Effectiveness”, IEEE Transactions on Computers, C-21, No 9, September, 1972.

[Gupt93] Gupta A., Kumar V., “Performance properties of large scale parallel systems”, Journal of Parallel and Distributed Computing, November 1993.

[Gust88] Gustafson J., “Reevaluating Amdahl´s Law”, Comm. Of the ACM, Volume 31 (1988), pp. 532-533.

[Gust92] Gustafson J. L., “The consequences of fixed time performance measurement”, Proceedings of the 25th Hawaii International Conference on System Sciences, Volume III, pp 113-124, 1992.

[Heer91] Heermann D. W., Burkitt A. N., "Parallel Algorithms in Computational Science", Springer-Verlag, 1991.

[Hoar85] Hoare C. A. R., "Communicating Sequential Processes", Pentice-Hall, 1985.

[Hwan93] Hwang K., “Advanced Computer Architecture: Paralelism, Scalability, Programability”, McGraw-Hill, 1993.

[Kung90] Kung H. T., “Heterogeneous Multicomputers” Carnegie Mellon Computer Science: A 25-Year Commemorative, R.F. Rashid ed., pp. 235-251. Reading, Mass.: Addison-Wesley, 1990.

[Kung91] Kung H. T., Sansom R., Schlick S., Steenkiste P., Arnould M., Bitz F. J., Christianson F., Cooper E. C., Menzilcioglu O., Ombres D., and Zill B., “Network-Based Multicomputers: An Emerging Parallel Architecture” Proc. Supercomputing '91, pp. 664-673. IEEE, Albequerque, Nov. 1991.

[Laws92] H. Lawson, "Parallel processing in industrial real time applications", Prentice Hall 1992.

[Leig92] Leighton F. T., “Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes”, Morgan Kaufmann Publishers, 1992.

[Logi94] “Transputer Toolset”, Logical System, 1994.

[Mill98] Miller R., Stout Q. F., “Algorithmic Techniques for Networks of Processors”, CRC Handbook of Algorithms and Theory of Computation, M. J. Atallah, ed, 1998.

[Mors94] Morse F., “Practical Parallel Computing”, AP Professional, 1994.

[Niel90] Nielsen K., “Ada in Distributed Real-Time Systems”, McGraw-Hill, 1990.

[Niga95] Nigam M., Sahni S., “Sorting n^2 Numbers on $n \times n$ Meshes”, IEEE Transactions on Parallel and Distributed Systems, Vol. 6, No. 12: December 1995, pp. 1221-1225

[Stee96] Steenkiste P., “Network-Based Multicomputers: A Practical Supercomputer Architecture”, IEEE Transactions on Parallel and Distributed Systems, Vol. 7, No. 8, August 1996, pp. 861-875

[SunX90] Sun X., Ni L. M., “Another view of parallel speedup”, Supercomputing ´90 Proceedings, pp 324-333, 1990.

[SunX95] Sun X., Zhu J.,“Performance Considerations of Shared Virtual Memory Machines”, Transactions on Parallel and Distribued Systems, Vol 6, No. 11: November 1995, pp. 1185-1194.

[Tine98] Tinetti F., De Giusti A., "Procesamiento Paralelo. Conceptos de Arquitectura y Algoritmos", Editorial Exacta, 1998.

[WuIC91] Wu I. C., Kung H. T., “Communication Complexity for Parallel Divide-and-Conquer” Proc. Symp. Foundations of Computer Science, pp. 151-162. San Juan, Oct. 1991.